

Contents

Comparisons and Ordering	1
Arithmetic	2
Boolean	3
Sequences	3
Shift And Rotate	5
Random Values	5
Debugging	5
Hints for Hardware Generation?	5

Comparisons and Ordering

Old types:

```
(==)  : {a}   (fin a) => (a,a) -> Bit
(!=)  : {a}   (fin a) => (a,a) -> Bit
(===) : {a b} (fin b) => (a -> b,a -> b) -> a -> Bit
(!==) : {a b} (fin b) => (a -> b,a -> b) -> a -> Bit

(<)   : {n} (fin n) => ([n],[n]) -> Bit
(>)   : {n} (fin n) => ([n],[n]) -> Bit
(<=)  : {n} (fin n) => ([n],[n]) -> Bit
(>=)  : {n} (fin n) => ([n],[n]) -> Bit

min   : {n} (fin n) => ([n],[n]) -> [n]
max   : {n} (fin n) => ([n],[n]) -> [n]
```

New types:

```
(==)  : {a}   (Cmp a) => a -> a -> Bit
(!=)  : {a}   (Cmp a) => a -> a -> Bit
(===) : {a,b} (Cmp b) => (a -> b) -> (a -> b) -> a -> Bit
(!==) : {a,b} (Cmp b) => (a -> b) -> (a -> b) -> a -> Bit

(<)   : {a} (Cmp a) => a -> a -> Bit
(>)   : {a} (Cmp a) => a -> a -> Bit
(<=)  : {a} (Cmp a) => a -> a -> Bit
(>=)  : {a} (Cmp a) => a -> a -> Bit
```

```

min    : {a} (Cmp a) => a -> a -> a
max    : {a} (Cmp a) => a -> a -> a

instance Cmp Bit
// No instance for functions.
instance (Cmp a, fin n) => Cmp [n] a
instance (Cmp a, Cmp b) => Cmp (a,b)
instance (Cmp a, Cmp b) => Cmp { x : a, y : b }

```

Arithmetic

Old types:

```

negate : {n a} (n >= 1) => [n]a -> [n]a
(+)    : {n a} ([n]a, [n]a) -> [n]a
(-)    : {n a} ([n]a, [n]a) -> [n]a
(*)    : {n a} ([n]a, [n]a) -> [n]a
(/)    : {n a} ([n]a, [n]a) -> [n]a
(%)    : {n a} ([n]a, [n]a) -> [n]a
(**)   : {n a} ([n]a, [n]a) -> [n]a

```

New types:

```

negate : {a} (Arith a) => a -> a
(+)    : {a} (Arith a) => a -> a -> a
(-)    : {a} (Arith a) => a -> a -> a
(*)    : {a} (Arith a) => a -> a -> a
(/)    : {a} (Arith a) => a -> a -> a
(%)    : {a} (Arith a) => a -> a -> a
(^^))  : {a} (Arith a) => a -> a -> a

// No instance for `Bit`.
instance (fin n)          => Arith ( [n] Bit)
instance (Arith a)       => Arith ( [n] a)
instance (Arith b)       => Arith (a -> b)
instance (Arith a, Arith b) => Arith (a,b)
instance (Arith a, Arith b) => Arith { x : a, y : b }

```

Note that because there is no instances for `Arith Bit` the top two instances do not actually overlap.

A corner case: unlike the old system, we'd also have to define `negate` at type 0. This makes sense, there is only one element of type `[0]`, so it is naturally its own inverse, thus `negate` should behave as the identity function.

Boolean

Old types:

```
False : Bit
True  : Bit

zero  : {a} a
(&)   : {a} (a,a) -> a
(|)   : {a} (a,a) -> a
(^)   : {a} (a,a) -> a
(~)   : {a} a -> a
```

New types:

```
False : Bit
True  : Bit

zero  : (Logic a) => a
(&&)  : (Logic a) => a -> a -> a
(||)  : (Logic a) => a -> a -> a
(^)   : (Logic a) => a -> a -> a
(~)   : (Logic a) => a -> a

// There are instances for all types,
// so we could potentially omit the constraints.
instance Logic Bit
instance Logic a      => Logic ([n] a)
instance Logic b      => Logic (a -> b)
instance (Logic a, Logic b) => Logic (a,b)
instance (Logic a, Logic b) => Logic { x : a, y : b }
```

Sequences

Old types:

```
width      : {n a m} (m >= width n) => [n]a -> [m]

join       : {n m a} [n] [m]a -> [n*m]a
split     : {n m a} [n*m]a -> [n] [m]a
splitBy   : {n m a} (n, [n*m]a) -> [n] [m]a
groupBy   : {n m a} (m, [n*m]a) -> [n] [m]a

(#)       : {n a m} (fin n) => ([n]a, [m]a) -> [n+m]a
```

```

tail      : {n a} [n+1]a -> [n]a
take     : {n m a} (fin n,m >= n) => (n,[n+m]a) -> [n]a
drop     : {n m a} (fin n,n >= n) => (n,[n+m]a) -> [m]a

reverse  : {n a} (fin n) => [n]a -> [n]a
transpose : {n m a} [n][m]a -> [m][n]a

(@)      : {n a m}                ([n]a,[m]) -> a
(@@)     : {n a m i}              ([n]a,[m][i]) -> [m]a
(!)      : {n a m} (fin n) => ([n]a,[m]) -> a
(!!)     : {n a m i} (fin n) => ([n]a,[m][i]) -> [m]a

```

New types:

```

length   : {n,a,m} (m >= width n) => [n]a -> [m]

join     : {parts,ench,a} (fin each) => [parts][each]a -> [parts * each]a
split    : {parts,each,a} (fin each) => [parts * each]a -> [parts][each]a

(#)      : {front,back,a} (fin front) => [front]a -> [back]a -> [front + back]a
splitAt  : {front,back,a} (fin front) => [front + back] a -> ([front] a, [back] a)

reverse  : {n,a} (fin n) => [n]a -> [n]a
transpose : {n,m,a} [n][m]a -> [m][n]a

(@)      : {n a m}                [n]a -> [m]    -> a
(@@)     : {n a m i}              [n]a -> [m][i] -> [m]a
(!)      : {n a m} (fin n) => [n]a -> [m]    -> a
(!!)     : {n a m i} (fin n) => [n]a -> [m][i] -> [m]a

```

// Abbreviations

```

splitBy n = split`{parts = n}
groupBy n = split`{each = n}
tail n    = splitAt`{front = 1}.2
take n    = splitAt`{front = n}.1
drop n    = splitAt`{front = n}.2

```

/* Also, `length` is not really needed:

```

length : {n,a,m} (m >= width n) => [n]a -> [m]
length _ = `n
*/

```

Shift And Rotate

Old types:

```
(<<<) : {n a m} (m >= lg2 n, fin n) => ([n]a, [m]) -> [n]a
(>>>) : {n a m} (m >= lg2 n, fin n) => ([n]a, [m]) -> [n]a
(<<<<) : {n a m} (m >= lg2 n, fin n) => ([n]a, [m]) -> [n]a
(>>>>) : {n a m} (m >= lg2 n, fin n) => ([n]a, [m]) -> [n]a
```

New types:

```
(<<<) : {n,a,m} (fin n, Logic a) => [n]a -> [m] -> [n]a
(>>>) : {n,a,m} (fin n, Logic a) => [n]a -> [m] -> [n]a
(<<<<) : {n,a,m} (fin n, Logic a) => [n]a -> [m] -> [n]a
(>>>>) : {n,a,m} (fin n, Logic a) => [n]a -> [m] -> [n]a
```

Random Values

Old types:

```
random : {a b} => [a] -> b
```

New types:

```
random : {a} => [32] -> a
```

Debugging

```
ASSERT      : {n a} (Bit, [n] [8], a) -> a
undefined   : {a} a
error       : {n a} [n] [8] -> a
trace       : {n a j} ([n] [8], a, j) -> j
```

Hints for Hardware Generation?

```
pipeline_stop : {a} a -> a
pipeline      : {n a} (fin n) => ([n], a) -> a

seq           : {n a} [n]a -> [n]a
par          : {n a} [n]a -> [n]a
reg          : {a} a -> a
const        : {a} a -> a
```